

Building a Neural Semantic Parser from a Domain Ontology

Jianpeng Cheng

University of Edinburgh

JIANPENG.CHENG@ED.AC.UK

Siva Reddy

Stanford University

SIVA.REDDY@CS.STANFORD.EDU

Mirella Lapata

University of Edinburgh

MLAP@INF.ED.AC.UK

Abstract

Semantic parsing is the task of converting natural language utterances into machine interpretable meaning representations which can be executed against a real-world environment such as a database. Scaling semantic parsing to arbitrary domains faces two interrelated challenges: obtaining broad coverage training data effectively and cheaply; and developing a model that generalizes to compositional utterances and complex intentions. We address these challenges with a framework which allows to elicit training data from a domain ontology and bootstrap a neural parser which recursively builds derivations of logical forms. In our framework meaning representations are described by **sequences of natural language templates**, where each template corresponds to a decomposed fragment of the underlying meaning representation. Although artificial, templates can be understood and paraphrased by humans to create natural utterances, resulting in parallel triples of utterances, meaning representations, and their decompositions. These allow us to train a neural semantic parser which learns to compose rules in deriving meaning representations. We crowdsource training data on six domains, covering both single-turn utterances which exhibit rich compositionality, and sequential utterances where a complex task is procedurally performed in steps. We then develop neural semantic parsers which perform such compositional tasks. In general, our approach allows to deploy neural semantic parsers quickly and cheaply from a given domain ontology.

1. Introduction

Semantic parsing has recently emerged as a key technology towards developing systems that understand natural language and enable interactions between humans and computers. A semantic parser converts natural language utterances into machine interpretable meaning representations, such as programs or logical forms. These representations can be executed against a real-world environment, such as navigating a database, comparing products, or controlling a robot (Kwiatkowski, Zettlemoyer, Goldwater, & Steedman, 2011; Liang, Jordan, & Klein, 2011; Wen, Gasic, Mrkšić, Su, Vandyke, & Young, 2015; Matuszek, Herbst, Zettlemoyer, & Fox, 2013). Table 1 shows examples of natural language utterances and their corresponding meaning representations in two application scenarios.

The development of semantic parsers faces two interrelated challenges, namely how to obtain training data efficiently and cheaply and how to handle compositional utterances and intentions. As far as the first challenge is concerned, semantic parsers have been mostly trained on data consisting of utterances paired with human-annotated meaning representations (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2005a; Wong & Mooney, 2006;

Task	Aggregated NL Utterances	Decomposed NL utterances	Meaning Representations
Querying a database	<i>Find all Chinese restaurants near me.</i>	<i>Find all Chinese restaurants near me.</i>	<code>r1 = filter_(find_all(restaurants), food.type, chinese)</code>
	<i>Find all Chinese restaurants near me. My maximum budget is 50\$ and list only those with restrooms.</i>	<i>Which ones cost no more than 50\$?</i> <i>And have restrooms?</i>	<code>r2 = filter_(r1, distance, 500m)</code> <code>r3 = filter_(r2, price, 50\$)</code> <code>r4 = filter_assertion(r3, has_restroom)</code>
Instructing a robot	<i>Place the kettle in the sink and fill it with water.</i>	<i>Place the kettle in the sink and fill it with water.</i>	<code>move(kettle, sink)</code> <code>toggle(sinkknob, on)</code>
	<i>Place the kettle under the tap and fill it with water. When it is filled, turn off the tap and heat the kettle with the stove, until the water is boiled.</i>	<i>Turn off the tap and heat the kettle with the stove.</i> <i>Wait until the water is boiled.</i>	<code>wait_until(fill)</code> <code>toggle(sinkknob, off)</code> <code>move(kettle, stove)</code> <code>toggle(stoveknob, on)</code> <code>wait_until(boil)</code> <code>toggle(stoveknob, off)</code>

Table 1: Examples of natural language (NL) utterances and corresponding meaning representations. Human intentions may be specified as a longer and more compositional utterance, or a sequence of inter-related short utterances.

Kwiatkowski, Zettlemoyer, Goldwater, & Steedman, 2010a; Dong & Lapata, 2016; Jia & Liang, 2016). Labeling such data is labor-intensive and error-prone: annotators must not only be trained with the knowledge about the logical language and the domain of interest, but also need to ensure every meaning representation they create actually matches the utterance semantics. Because of this reason, it takes much effort to develop neural semantic parsers for new domains where no training data exists; and for domains whose ontology changes frequently (training data needs to be updated accordingly). This annotation challenge becomes more obvious when human intentions become complex. Table 1 shows examples of complex intentions expressed as a single compositional utterance or as a sequence of simpler utterances. In either case, writing down the correct meaning representations is not trivial.

As for the second challenge, traditional semantic parsers (Zettlemoyer & Collins, 2005b; Wong & Mooney, 2006; Kwiatkowski et al., 2010a; Kwiatkowski, Choi, Artzi, & Zettlemoyer, 2013; Berant, Chou, Frostig, & Liang, 2013a) adopt a domain-specific grammar, a trainable model, and a parsing algorithm. The grammar defines the space of possible derivations from an utterance to a logical form, and the model together with the parsing algorithm finds the most likely derivation. A chart-based parsing algorithm is commonly used to parse an utterance in polynomial time. Recent advances in neural networks have spurred new interest in reformulating semantic parsing as a sequence-to-sequence learning problem (Bahdanau, Cho, & Bengio, 2015). Such neural semantic parsers (Dong & Lapata, 2016; Jia & Liang, 2016) parse utterances in linear time, while reducing the need for grammar and feature engineering. But this modeling flexibility comes at a cost since it is less possible

to interpret how meaning composition is performed—because meaning representations are treated as strings rather than structured objects (e.g., trees or graphs). Such knowledge plays a critical role in building more generalizable neural semantic parsers, especially in the face of sparse data. For this reason, subsequent development of neural semantic parsers focuses on **sequence-to-action** models which handle meaning composition explicitly (Cheng, Reddy, Saraswat, & Lapata, 2017a, 2017b; Yin & Neubig, 2018; Gupta, Shah, Mohit, Kumar, & Lewis, 2018). Besides, most previous work on neural semantic parsing has focused on isolated utterances, ignoring the fact that some intentions are more likely to be expressed through a sequence of co-referring utterances (see Table 1).

Our first contribution in this paper is a method for eliciting neural semantic parsing data which expresses complex intentions, from a domain ontology. Our approach builds on the work of Wang et al. (2015) who advocate crowd-sourcing as a way of mitigating the paucity of semantic parsing datasets. Their basic idea is to use **a synchronous grammar to generate meaning representations paired with artificial utterances** which crowdworkers are asked to paraphrase into more natural sounding utterances. For example, `argmin(food_type(Thai food), distance)` is deterministically mapped to “*restaurants with smallest distance where food type is thai*”, which will be later paraphrased by crowdworkers into e.g., “*nearest restaurants serving thai food*”. However, we experimentally found out the readability of these artificial utterances decreases when the complexity of the task increases. As an example, the artificial utterance “*restaurants where food type is food type of kfc which has minimum price*” is rather difficult to interpret due to the attachment ambiguity caused by relative clause “*which has minimum price*”—it is unclear whether it modifies *kfc*, *food*, or *restaurants*. The approach primarily targets utterances exhibiting shallow compositionality often with two predicates and entities (Wang et al., 2015), thereby avoiding ambiguities arising from complex intentions. Instead of representing the meaning of a task with a single artificial description, our approach represents it as a sequence of inter-related templates, where each template corresponds to a decomposed fragment of meaning. So, the example above would be represented with templates “**Result₁ = find food type kfc**” and “**Result₂ = find restaurants with food type Result₁**” and “**Result₃ = find Result₂ with minimum price**”. Iyyer et al. (2017) show that decomposed utterances can capture higher levels of compositionality in practice. Since templates correspond to specific parts of the meaning representation, they are easier to understand by crowdworkers compared to more elaborate artificial descriptions. Furthermore, the templates allow us to flexibly crowdsource two different types of data to study human intentions expressed in different ways: we can obtain individual utterances which correspond to more compositional meaning representations, or a sequence of inter-related utterances, depending on whether participants are asked to summarize or paraphrase the templates.

The second contribution of this paper is a neural semantic parsing framework which leverages the annotations (in the form of template sequences) elicited by the above method and the ability of recurrent neural networks to model compositionality (Dyer, Kuncoro, Ballesteros, & Smith, 2016; Cai, Shin, & Song, 2017). Our model is based on the fact that **a sequence of templates encodes the rules whose application in recursive order yields the**

final meaning representation. The parser is thus trained to predict derivations¹, obtaining meaning representations by composing the rules. Specifically, we adopt a transition action-based approach which handles the generation of domain-general and domain-specific rules in a unified way, with constraints ensuring the rules can be composed smoothly. An important challenge in semantic parsing is tackling mismatches between natural language and representation language. For example, both utterances “*cheapest restaurants*” and “*restaurants with smallest price rating*” trigger an **argmax** rule despite expressing the same information need in different ways. We resolve this challenge **with a neural attention mechanism, which learns a soft mapping between natural language and meaning representation language.** The neural semantic parser is also designed to handle sequential utterances² which involve co-reference.

We conduct a wide range of experiments to evaluate the proposed framework. As a testbed, we elicit annotations for database querying tasks involving compositional user intentions. Using crowdsourcing, various templates underlying computer-generated meaning representations are labeled with either single or sequential utterances. We crowdsource data covering six domains and provide detailed analysis on the annotations. We then use the data to train a neural semantic parser which handles compositionality and co-reference. Overall, we advocate an end-to-end solution which allows to build neural semantic parsers quickly and cheaply, starting with a domain ontology.

The remainder of this paper is structured as follows. Section 2 discusses related work of semantic parsing. Section 3 introduces our data elicitation method while Section 4 presents the neural semantic parsing model. Section 5 highlights our experimental results. Finally, Section 6 concludes the paper.

2. Related Work

Early semantic parsing systems are hard-coded to answer questions in constrained domains. The LUNAR system (Winograd, 1972) were designed to handle questions about moon rocks using a large database. It converts queries into programs by mapping syntactic fragments to semantic units. Another example is the SHRDLU system (Winograd, 1972) which launches dialogs between the user and the system-simulated robot to manipulate simple objects on a table. Central to these systems is the idea of expressing words and sentences as computer programs, and the execution of programs corresponds to the reasoning of meanings. However, the development of these systems require a large deal of domain-specific knowledge and engineering.

In reaction to these problems in 1970s, the focus of semantic parsing research shifted from rule-based method to empirical or statistical methods, where data and machine learning plays an important role. Statistical semantic parsers typically consist of three key components: a grammar, a trainable model, and a parsing algorithm. The grammar defines

-
1. In this work, a derivation tree refers to a parse tree that graphically represents the semantic information of how a meaning representation is derived from a context-free grammar, which does not rely on tokens in the corresponding utterance—this is slightly different from the definition of derivation in a chart parser.
 2. This work studies parsing sequential utterances in a non-dialog setup: the model does not involve decision making on the optimum strategy of responding to each input utterance. Instead, it simply outputs the execution result of the obtained meaning representation.

the space of derivations from utterances to meaning representations, and the model together with the parsing algorithm find the most likely derivation. An example of early statistical semantic parser is the CHILL system (Zelle & Mooney, 1996) based on inductive logic programming (ILP). The system uses ILP to learn control rules for a shift-reduce parser. To train and evaluate their system, Zelle and Mooney (1996) created the GEOQUERY dataset which contains 880 queries to a US geography database. These queries are paired with annotated meaning representations in Prolog.

Until early 2000, semantic parsing research mainly focused on restricted domains. Besides GEOQUERY, commonly used datasets are ROBOCUP for coaching advice to soccer agents (Kitano, Tambe, Stone, Veloso, Coradeschi, Osawa, Matsubara, Noda, & Asada, 1997), and ATIS for air travel information service (Price, 1990). At that time, statistical approaches for parsing domains-specific context-free grammars have been largely explored. For example, Kate and Mooney (2006) propose KRISP, which induces context-free grammar rules that generate meaning representations, and uses kernel SVM to score derivations. Ge and Mooney (2005) propose SCISSOR, which employs an integrated statistical parser to produce a semantically augmented parse tree. Each non-terminal node in the tree has both a syntactic and a semantic label, from which the final meaning representation can be derived. The WASP system proposed by Wong and Mooney (2007) learns synchronous context free grammars that generate utterances and meaning representations. Parsing is achieved by finding the most probable derivation that leads to the utterance and recovering the meaning representation with synchronous rules. Lu et al. (2008) proposes a generative model for utterances and meaning representations. Similar to Ge and Mooney (2005), they define hybrid trees whose nodes include both words and meaning representation tokens. Training is performed with the EM algorithm. The model, especially the generative process, was extend by Kim and Mooney (2010) to learn from ambiguous supervisions.

The next breakthrough came with the work of Zettlemoyer and Collins (2005b), who introduced CCG in semantic parsing. Their probabilistic CCG grammars can deal with long range dependencies and construct non-projective meaning representations. A great deal of work follows Zettlemoyer and Collins (2005b) but focuses on more fine-grained problems such as grammar induction and lexicon learning (Kwiatkowski, Zettlemoyer, Goldwater, & Steedman, 2010b; Kwiatkowski et al., 2011; Krishnamurthy & Mitchell, 2012; Artzi, Das, & Petrov, 2014; Krishnamurthy & Mitchell, 2015; Krishnamurthy, 2016; Gardner & Krishnamurthy, 2017) or using less supervision (Artzi & Zettlemoyer, 2013a; Reddy, Lapata, & Steedman, 2014). As a common paradigm, the class of work first generates candidate derivations to meaning representations governed by the grammar. These candidates derivations are scored by a trainable model which can take the form of a structured perceptron (Zettlemoyer & Collins, 2007) or a log-linear model (Zettlemoyer & Collins, 2005b). Training updates model parameters such that good derivations obtain higher scores. During inference, a CKY-style chart parsing algorithm is used to predict the most likely derivation for an utterance. Another class of work follows similar paradigm but use lambda DCS as the semantic formalism (Berant, Chou, Frostig, & Liang, 2013b; Berant & Liang, 2014, 2015). Other interesting work includes joint semantic parsing and grounding (Kwiatkowski et al., 2013), parsing context-dependent queries (Artzi & Zettlemoyer, 2013b; Long, Pasupat, & Liang, 2016), and converting dependency trees to meaning representations (Reddy,

Täckström, Collins, Kwiatkowski, Das, Steedman, & Lapata, 2016; Reddy, Täckström, Petrov, Steedman, & Lapata, 2017).

With recent advances in neural networks and deep learning, there is a trend of reformulating semantic parsing as a machine translation problem, which converts a natural language sequence into a programming language consequence. The idea was not novel and has been previously studied with statistical machine translation approaches. For example, both Wong and Mooney (2006) and Andreas et al. (2013) developed word-alignment based translation models for parsing the GEOQUERY dataset. However, the task setup is important to be revisited since recurrent neural networks have been shown to be extremely useful in context modeling and sequence generation (Bahdanau et al., 2015). Following this direction, Dong and Lapata (2016) and Jia and Liang (2016) developed neural semantic parsers which treat semantic parsing as a sequence-to-sequence learning problem. Surprisingly, the approach has been proven effectively on even the small GEOQUERY dataset. Jia and Liang (2016) further introduces a data augmentation approach which bootstraps a synchronous grammar from existing data and generates artificial examples as extra training data. State of the art result on GEOQUERY dataset was obtained with this approach. Subsequent work of Kočiský et al. (2016) attempts to explore the meaning representation space with a generative autoencoder. They bootstrap a probabilistic monolingual grammar for meaning representations, from which unseen meaning representations can be sampled. These samples are used as semi-supervised training data to the autoencoder. Other related work extends the vanilla sequence to sequence model in various ways, such as employing two encoder-decoders for coarse to fine decoding (Dong & Lapata, 2018), handling multiple tasks with a shared encoder (Fan, Monti, Mathias, & Dreyer, 2017), parsing cross-domain queries (Herzig & Berant, 2017) and context-dependent queries (Suhr, Iyer, & Artzi, 2018), and applying the model to other formalisms such as AMR (Konstas, Iyer, Yatskar, Choi, & Zettlemoyer, 2017) and SQL (Zhong, Xiong, & Socher, 2017; Xu, Liu, & Song, 2017).

The fact that meaning representations have a syntactic structure has motivated more recent work on exploring structured neural decoders to generate tree or graph structures, and grammar constrained decoders to make sure the outputs are meaningful and executable. For example, Yin and Neubig (2017) generate abstract syntax trees for source code with a grammar constrained neural decoder. Krishnamurthy et al. (2017) also introduce a neural semantic parser which decodes rules of a grammar to obtain well-typed meaning representations. Cheng et al. (2017a, 2017b), Yin and Neubig (2018), Gupta et al. (2018), Chen et al. (2018) all employ neural sequence-to-action models to generate structured meaning representations.

3. Data Elicitation

As shown above, most work on semantic parsing (including neural semantic parsing) has used existing datasets with annotated utterance-meaning representation pairs (Zelle & Mooney, 1996; Ge & Mooney, 2005; Kate & Mooney, 2006; Kate, Wong, & Mooney, 2005; Wong & Mooney, 2006; Lu et al., 2008; Kwiatkowski et al., 2010a; Dong & Lapata, 2016; Jia & Liang, 2016). In contrast, our work focuses on a practical scenario when one wants to develop a neural semantic parser from a new domain ontology: there exists no prior training data but the expected utterances can be arbitrarily compositional. Two challenges

arise here for data collection: 1) although training data in form of utterance-meaning representation pairs provides an effective training signal, their annotation is labor intensive and error-prone. 2) although utterances (e.g., usage logs if exist) can be sampled for the given domain, it is not easy to ensure this data covers a broad range of compositional patterns.

In this section, we detail a data elicitation method which collects training data of neural semantic parsers cheaply and effectively, for both single-turn and sequential utterances. The idea is to use a computer program to generate valid meaning representations based on the domain ontology. These meaning representations are mapped to an artificial human language, which can be understood by annotators to create utterances. In summary, annotators are generating utterances for meaning representations, instead of generating meaning representations for utterances.

3.1 Decomposition of Meaning Representations

Our approach follows Wang et al. (2015) to decompose meaning representations into various constructs. The decomposition allows us to build a program which generates meaning representations with broad coverage; and explicitly model the generation process during parsing. Throughout this paper, we exemplify our approach with a database querying task. Specifically, our meaning representations are written in lambda expressions representing rules and variables in a computer program that queries a database.

The first construct of meaning representations are *domain-general* rules stemming from the formal language used by the semantic parser. In Table 2 we provide examples of domain-general rules represented as lambda expressions. These rules specifying various functionalities such as looking up a column in the database, counting, aggregation, and filtering by condition. They are generic, apply across domains, and relevant to the database querying task. The second construct of meaning representations are *domain-specific* rules which generate domain-specific predicates or entities. Table 3 shows example predicates and entities (which are represented as variables in the formal language) from the restaurant domain. We are assuming access to a domain-specific ontology which covers binary predicates for properties (e.g., `custom_rating`), unary predicates for assertions (e.g., `open_now`), and entities (e.g., `restaurant.kfc`).

The third construct of meaning representations captures complex human intentions with co-referential variables, which establish anaphoric links between meaning representations (antecedents and consequents). To model co-reference, we adopt the notions of discourse referents (DRs) and discourse entities (DEs) which are widespread in discourse representation theories (Webber, 1978; Kamp & Reyle, 2013). DRs are referential expressions appearing in utterances which denote DEs, i.e., mental entities in the speakers model of discourse. Co-referential variables imply that DEs in the antecedent and consequent refer to the same real-world entity (or entities) which we obtain from the execution of the antecedent. In corresponding natural language utterances, co-reference manifests itself by the explicit or implicit occurrence of a pronoun or a DR (e.g., a definite noun phrase) in the consequent. The main principle in determining whether DRs co-refer is that it must be possible to infer their relation from the dialogue context alone, without using world knowledge. Example (1) below shows various continuations of the utterance *Which restaurants serve thai food?* involving explicit co-reference (i.e., *of those* in (1-a)), implicit co-reference

Category	Domain-general Rules	Description and Evaluation
LookupKey	$\lambda s: (\text{lookupKey } (\text{var } s))$	Looks for the entire set of s
LookupValue	$\lambda p \lambda s: (\text{lookupValue } (\text{var } s) (\text{var } p))$	Looks for specific property p of entity s
Filter(property)	$\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) = (\text{var } v))$	Looks for subset of s whose property p equates to some value v
Filter(assertion)	$\lambda s \lambda p: (\text{filter } (\text{var } s) (\text{var } p) = \text{true})$	Looks for subset of s which satisfies condition p
Count	$\lambda s: (\text{size } (\text{var } s))$	Computes total number of elements in set s
Sum	$\lambda s: (\text{sum } (\text{var } s))$	Computes total sum of numerical elements in set s
Comparative ($<$)	$\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) < (\text{var } v))$	Looks for subset of s whose numeric property p is smaller than some numeric value v
Comparative (\leq)	$\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) \leq (\text{var } v))$	Looks for subset of s whose numeric property p is smaller than or equal to some numeric value v
Comparative ($>$)	$\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) > (\text{var } v))$	Looks for subset of s whose numeric property p is larger than some numeric value v
Comparative (\geq)	$\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) \geq (\text{var } v))$	Looks for subset of s whose numeric property p is larger than or equal to some numeric value v
CountComparative ($<$)	$\lambda s \lambda p \lambda v: ((\text{var } s) (\text{size } (\text{var } p)) < (\text{var } v))$	Looks for subset of s where the cardinality of property p is smaller than some numeric value v
CountComparative (\leq)	$\lambda s \lambda p \lambda v: ((\text{var } s) (\text{size } (\text{var } p)) \leq (\text{var } v))$	Looks for subset of s where the cardinality of property p is smaller than or equal to some numeric value v
CountComparative ($>$)	$\lambda s \lambda p \lambda v: ((\text{var } s) (\text{size } (\text{var } p)) > (\text{var } v))$	Looks for subset of s where the cardinality of property p is larger than some numeric value v
CountComparative (\geq)	$\lambda s \lambda p \lambda v: ((\text{var } s) (\text{size } (\text{var } p)) \geq (\text{var } v))$	Looks for subset of s where the cardinality of property p is larger than or equal to some numeric value v
Superlative (min)	$\lambda s \lambda p: ((\text{var } s) \text{argmin } (\text{var } p))$	Looks for subset of s whose numeric property p is smallest
Superlative (max)	$\lambda s \lambda p: ((\text{var } s) \text{argmax } (\text{var } p))$	Looks for subset of s whose numeric property p is largest
CountSuperlative (min)	$\lambda s \lambda p: ((\text{var } s) \text{argmin } (\text{size } (\text{var } p)))$	Looks subset of s where the cardinality of property p is smallest
CountSuperlative (max)	$\lambda s \lambda p: ((\text{var } s) \text{argmax } (\text{size } (\text{var } p)))$	Looks for subset of s where the cardinality of property p is largest

Table 2: Domain-general rules (and their descriptions) used to define meaning representations in our experiments.

Category	Predicates and Entities	Description
BinaryPredicate	custom_rating price_rating distance num_reviews location cuisine open_time	Overall rating from customers Price rating from customers Distance of the restaurant Number of reviews from customers Location of the restaurant Type of food served by the restaurant Opening time of the restaurant
UnaryPredicate	open_now take_away reservation credit_card waiter delivery kids groups	Is the restaurant opening now? Does the restaurant offer take-away? Does the restaurant accept reservations? Does the restaurant accept credit cards? Does the restaurant have waiter service? Does the restaurant offer delivery? Is the restaurant suitable for kids? Is the restaurant suitable for groups?
Entity	restaurant.kfc location.oxford_street	KFC Oxford Street

Table 3: Domain-specific predicates and entities from a restaurant domain, covering binary predicates (properties), unary predicates (assertions), and entities.

Category	Description
Coref	Refers to a single antecedent
Union_coref	Refers to the union of two antecedents
Intersection_coref	Refers to the intersection of two antecedents

Table 4: Co-reference variables and their descriptions.

(see (1-b)), and co-reference via a definite expression (i.e., *thai restaurants* in (1-c)). All these co-references are meant to be represented by co-referential variables on the meaning representation side. In this work, we consider three types of co-referential variables shown in Table 4 which can be used in place of type-matched, domain-specific entities to construct meaning representations.

- (1) Which restaurants serve thai food?
 - a. Of those which ones are nearest to me?
 - b. Nearest to me?
 - c. I mean thai restaurants nearest to me.

3.2 Mapping Rules to Human Language

The central idea behind our data elicitation method is to convert meaning representations to artificial descriptions, by mapping the various rules used to construct meaning represen-

Category	NL Templates
LookupKey	<i>find all of \$s</i>
LookupValue	<i>find \$p of \$s</i>
Filter(property)	<i>find \$s where \$p is \$v</i>
Filter(assertion)	<i>find \$s which satisfies \$p</i>
Count	<i>count number of elements in \$s</i>
Sum	<i>sum all elements in \$s</i>
Comparative(<)	<i>find \$s with \$p < \$v</i>
Comparative(>)	<i>find \$s with \$p > \$v</i>
Comparative(\leq)	<i>find \$s with \$p \leq \$v</i>
Comparative(\geq)	<i>find \$s with \$p \geq \$v</i>
CountComparative(<)	<i>find \$s with number of \$p < \$v</i>
CountComparative(>)	<i>find \$s with number of \$p > \$v</i>
CountComparative(\leq)	<i>find \$s with number of \$p \leq \$v</i>
CountComparative(\geq)	<i>find \$s with number of \$p \geq \$v</i>
Superlative(min)	<i>find \$s with smallest \$p</i>
Superlative(max)	<i>find \$s with largest \$p</i>
CountSuperlative(min)	<i>find \$s with smallest number of \$p</i>
CountSuperlative(max)	<i>find \$s with largest number of \$p</i>

Table 5: Domain-general rules are associated with natural language (NL) templates specified by our framework.

tations to human language. The mapping enables the data elicitation procedures which will be described in Section 3.3.

Our method maps domain-general rules onto natural language templates with missing entries. Each template describes the functionality of a rule, while missing entries specify variables required by the rule. Table 5 displays the list of domain-general rules we use to query a database. The corresponding templates are described in such a way that can be understood by annotators who have no knowledge of the underlying meaning representation. Different from the more natural language descriptions shown in Table 2, templates are human readable formal descriptions which are deterministically mapped from meaning representations (see the right column in the table).

Domain-specific variables are mapped to natural language phrases with a lexicon specified by a domain manager. This lexicon is the only resource we ask domain managers to provide, for the purposes of describing the domain ontology. Note that natural language descriptions are important in cases where domain-specific predicates or entities are not verbalized (for example a predicate may be simply represented as an index `m.001` in the database). Such descriptions must be provided to annotators to allow for basic understanding, and to enable the paraphrasing task. However, we do not use this lexicon for building a semantic parser. Table 6 displays a lexicon for the restaurant domain. Natural language descriptions of predicates and entities are used to instantiate templates.

Finally, for co-referential variables, we directly assign an index (e.g., `Result1`, `Result2`) to each fragment of meaning representation, and use this index as the value of co-referential variables when they are used to instantiate templates.

Database Predicates/Entities	NL Expressions
custom_rating	customer rating
price_rating	price rating
distance	distance
num_reviews	number of customer reviews
location	location
cuisine	cuisine
open_time	opening time
open_now	opens now
take_away	offers take-away
reservation	takes reservations
credit_card	accepts credit cards
waiter	has waiter service
delivery	offers delivery
kids	suitable for kids
groups	suitable for groups
restaurant.kfc	KFC
location.oxford.street	Oxford Street

Table 6: Examples of domain-specific lexicon and corresponding natural language (NL) expressions for the restaurant domain.

3.3 Data Elicitation for Single-turn Utterances

We are now ready to describe our data elicitation procedures. As mentioned earlier, our goal is to collect utterance-meaning representation pairs which represent compositional intentions. One intention can be expressed within a single utterance, or a sequence of utterances. We first discuss the data elicitation procedures for single-turn utterances and then explain how it can be straightforwardly extended to the sequential scenario.

Step 1: Generating Meaning Representations We use a context-free grammar and a computer program to generate meaning representations by sampling domain-general and specific rules. Generation is performed in a bottom-up manner, so that larger pieces of meaning can be constructed from smaller ones. The application of each domain-general rule takes an expected amount (i.e. as defined by the grammar) of domain-specific or co-referential variables, and results in a new piece of meaning. Such bottom-up generation ensures the quality of meaning representations: the program can check whether the generation process should continue with denotational clues. Table 7 shows an example of the generation process.

Note that we do not consider complex co-referential structures between meaning representations in the single-turn case ³ and restrict the derivation to be tree-structured. Co-referential variables can be eliminated by replacing them with corresponding antecedent meaning representations.

3. In most of the cases, the next meaning representation co-refers to the previous one: $Result_2 = find\ Result_1\ with\ smallest\ distance$

1.	<p>Bottom-up construction of meaning representations (done by framework)</p> <p>Domain-general rule $\lambda s: (\text{lookupKey } (\text{var } s))$ is instantiated with domain-specific variable <code>type.restaurant</code> to obtain the first piece of meaning representation:</p> $\text{Result}_1 = (\text{lookupKey } (\text{type.restaurant}))$ <p>For the second piece of meaning representation. domain-general rule $\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) = (\text{var } v))$ is instantiated with domain-specific and co-referential variables <code>Result₁</code>, <code>rel.cuisine</code>, and <code>cuisine.thai</code>:</p> $\text{Result}_2 = (\text{filter } (\text{Result}_1) (\text{rel.cuisine}) = (\text{cuisine.thai}))$ <p>For the third piece of meaning representation, domain-general rule $\lambda s: (\text{lookupValue } (\text{var } s) (\text{var } p))$ is instantiated with domain-specific variables <code>restaurant.kfc</code> and <code>rel.distance</code>:</p> $\text{Result}_3 = (\text{lookupValue } (\text{restaurant.kfc}) (\text{rel.distance}))$ <p>The final piece of meaning representation is constructed with domain-general rule $\lambda s \lambda p \lambda v: (\text{filter } (\text{var } s) (\text{var } p) < (\text{var } v))$ and domain-specific variables <code>Result₂</code>, <code>rel.distance</code>, and <code>Result₃</code>:</p> $\text{Result}_4 = (\text{filter } (\text{Result}_2) (\text{rel.distance}) < (\text{Result}_3))$
2.	<p>Each piece of meaning representation is converted to a canonical representation described by templates. Templates associated with domain-general rules are instantiated with domain-specific and co-referential variables (shown in brackets):</p> $\begin{aligned} \text{Result}_1 &= \text{find all } [\text{restaurants}] \\ \text{Result}_2 &= \text{find } [\text{Result}_1] \text{ where } [\text{cuisine}] \text{ is } [\text{Thai}] \\ \text{Result}_3 &= \text{find } [\text{distance}] \text{ of } [\text{KFC}] \\ \text{Result}_4 &= \text{find } [\text{Result}_2] \text{ with } [\text{distance}] < [\text{Result}_3] \end{aligned}$
3.	<p>The templates are displayed to the annotators who summarize them into an utterance:</p> <p><i>Which restaurant has Thai food and is closer to me than KFC?</i></p>

Table 7: Example of our data elicitation procedures for single-turn utterances paired with meaning representations.

Also note that grammar constraints are enforced by a program validator to ensure that the meaning representations are syntactically valid (i.e., variables are type-checked) and semantically correct (i.e. no rules logically entail or contradict each other). For example, if the previous meaning representation applies a **Count** rule which returns a number, the next rule cannot be **LookupKey** since the expected argument is a database column instead of a number. If a **Filter** rule is applied to include only restaurants within 500 meters, it

does not make sense to use a subsequent **Filter** rule to look for restaurants which are more than 1 kilometers away.

Step 2: Converting Meaning Representations to Templates For each meaning representation fragment in the bottom-up derivation, we look up the corresponding template underlying the domain-general rule that was used to construct it. The template has missing entries, which are instantiated with domain-specific predicates (e.g., `rel.distance`), entities (e.g., `cuisine.thai`) or referents to previous templates (e.g., `Result1`). In the end, we obtain a sequence of instantiated templates which describe the intention of the final meaning representation.

Step 3: Annotating Utterances The instantiated templates are displayed to annotators, who are asked to create a corresponding utterance. The collected single-turn utterance does not have to be a single sentence, it can consist of a few sentences or clauses. We set no restrictions on the format of expression. As a result, we obtain single-turn utterances paired with meaning representations.

3.4 Data Elicitation for Sequential Utterances

We next discuss how the data elicitation method can be slightly modified to handle the collection of sequential utterances. As described in Section 3.1, sequential utterances can potentially exhibit a rich class of co-reference phenomena, either explicit or implicit. These phenomena are indicated by co-reference variables in corresponding meaning representations. We start by analyzing various co-reference structures arising in sequential utterance/meaning representations. We then proceed to elicit data reflecting these structures.

We follow an ad-hoc, inductive approach which enumerates co-reference structures in three consecutive meaning representations as a base case, from which more complex co-reference structures can be constructed. When the interpretation of meaning representation M2 (with corresponding utterance Q2 and template R2) depends on meaning representation M1 (with corresponding utterance Q1 and template R1), we call M2 (and Q2, R2) the consequent, and M1 (and Q1, R1) the antecedent. Co-reference structures in three consecutive utterance/meaning representations are shown in Figure 1 and described in more detail below.

Exploitation refers to chain-structured anaphoric links, where the consequent of the previous meaning representation is the antecedent of the next. For example, this happens when a user queries a database with incremental constraints. As a result, the next utterance consistently uses the denotation of the previous one.

Exploration refers to branch-structured anaphoric links, where an antecedent has two consequents. For example, this happens when a user explores different options or constraints related to the same antecedent.

Merging refers to the inverse of branch-structured anaphoric links, where a consequent has two antecedents. For example, this happens when a user combines the denotations of two or more utterances with union or intersection. The combined denotation set is then used in a subsequent utterance.

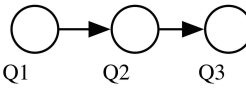
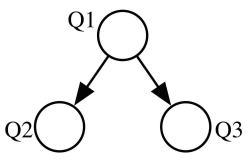
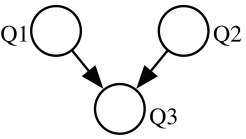
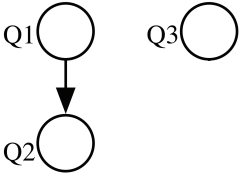
Exploitation		
	Q1: <i>restaurants in oxford street?</i> Q2: <i>which cost less than 50?</i> Q3: <i>with car parking?</i>	$R_1 = \text{find restaurants where location is oxford street}$ $R_2 = \text{find } [R_1] \text{ where price} < 50$ $R_3 = \text{find } [R_2] \text{ which has car parking}$
Exploration		
	Q1: <i>restaurants in oxford street</i> Q2: <i>with chinese food?</i> Q3: <i>oxford street restaurants with thai food?</i>	$\text{Result}_1 = \text{find restaurants where location is oxford street}$ $R_2 = \text{find } [R_1] \text{ where food type is chinese}$ $R_3 = \text{find } [R_1] \text{ where food type is thai}$
Merging		
	Q1: <i>find chinese restaurants.</i> Q2: <i>thai restaurants?</i> Q3: <i>chinese or thai restaurants with car parking?</i>	$R_1 = \text{find restaurants where food type is chinese}$ $R_2 = \text{find restaurants where food type is thai}$ $R_2 = \text{find } [R_1] \text{ and } [R_2] \text{ which has car parking}$
Unrelated		
	Q1: <i>which restaurants serve chinese food?</i> Q2: <i>those in oxford street?</i> Q3: <i>which restaurants serve thai food?</i>	$R_1 = \text{find restaurants where food type is chinese}$ $R_2 = \text{find } [R_1] \text{ where location is oxford street}$ $R_3 = \text{find restaurants where food type is thai}$

Figure 1: Coreference structures in three consecutive meaning representations and corresponding utterances. They are represented as nodes; edges are drawn between co-referring utterances; antecedents are nodes with outgoing edges while consequents are nodes with incoming edges.

Unrelated refers to two unconnected co-reference structures observed in a user session. For example, this happens when a user specifies two independent intentions in the same session.

We expect the four categories mentioned above to cover a majority of co-reference structures in sequential utterance/meaning representations. However, there exist meanings which they fail to represent or construct. An example is the following sequential utterances: “*find restaurants in oxford street*”, “*find restaurants in bond street*”, “*which of them serve chinese food*”, and finally “*how about those in oxford street*”. Note that the purpose of this work is not to formalize all possible co-reference structures.

In the following, we present our data elicitation method for sequential utterances which is also illustrated in Table 8.

Step1: Generating Meaning Representations As in the single-turn case, we use a context-free grammar and a computer program to generate meaning representations by sampling domain-general and specific rules. Again, the generation is performed in a bottom-up

1.	<p>Bottom-up construction of meaning representations (done by framework)</p> <p>Two domain-general rules $\lambda s:(\text{lookupKey } (\text{var } s))$ and $\lambda s\lambda p\lambda v:(\text{filter } (\text{var } s) (\text{var } p) = (\text{var } v))$ are instantiated with domain-specific variables <code>type.restaurant</code>, <code>rel.cuisine</code>, and <code>cuisine.thai</code> to obtain the first piece of meaning representation:</p> $\text{Result}_1 = (\text{filter } (\text{lookupKey } (\text{type.restaurant})) (\text{rel.cuisine}) = (\text{cuisine.thai}))$ <p>For the second piece of meaning representation, domain-general rule $\lambda s:(\text{lookupValue } (\text{var } s) (\text{var } p))$ is instantiated with domain-specific variables <code>restaurant.kfc</code> and <code>rel.distance</code>:</p> $\text{Result}_2 = (\text{lookupValue } (\text{restaurant.kfc}) (\text{rel.distance}))$ <p>For the third piece of meaning representation, domain-general rule $\lambda s\lambda p\lambda v:((\text{var } s) \text{ min } (\text{var } p))$ is instantiated with domain-specific variables <code>Result₁</code> and <code>rel.price</code>:</p> $\text{Result}_3 = ((\text{Result}_1) \text{ argmin } (\text{rel.price}))$ <p>The final piece of meaning representation is constructed with the domain-general rule $\lambda s\lambda p\lambda v:(\text{filter } (\text{var } s) (\text{var } p) < (\text{var } v))$ and domain-specific variables <code>Result₃</code>, <code>rel.distance</code>, and <code>Result₂</code>:</p> $\text{Result}_4 = (\text{filter } (\text{Result}_3) (\text{rel.distance}) < (\text{Result}_2))$
2.	<p>Each piece of meaning representation is converted to a canonical representation described by templates. Templates associated with domain-general rules are instantiated with domain-specific and co-referential variables (shown in brackets):</p> <p><i>Result₁ = find [restaurants] where [cuisine] is [Thai]</i> <i>Result₂ = find [distance] of [KFC]</i> <i>Result₃ = find [Result₁ with smallest] [price rating]</i> <i>Result₄ = find [Result₃] with [distance] < [Result₂]</i></p>
3.	<p>The templates are displayed to annotators who summarize them into an utterance:</p> <p style="text-align: center;"><i>Show me Thai restaurants.</i> <i>How far is KFC?</i> <i>Which Thai restaurants are cheapest?</i> <i>Which of these are closer to me than KFC?</i></p>

Table 8: Example of our data elicitation procedures for sequential utterances paired with meaning representations.

manner, so that larger pieces of meaning representation can be constructed from smaller ones. Each domain-general rule is instantiated with domain-specific or co-referential variables and results in a new piece of meaning. Different from the single-turn case, we allow

each meaning representation to be constructed with one to three domain-general rules, so as to have some basic level of compositionality. Furthermore, we consider a richer class of co-reference structures in-between meaning representations (in the single-turn case, the co-reference between meaning representations is mostly chain-structured).

We use the three terminal-level rules introduced in Table 4 to generate co-referential variables. For each co-referential variable, we additionally generate its value—which is selected from the index list of previously generated meaning representations. To recap, we assign an index to each meaning representation (e.g., `Result1`, `Result2`) and this index is used to denote the value of a co-referential variable. Similar to the single-turn case, grammar constrains are enforced by a validator to ensure the meaning representations are syntactically and semantically valid (e.g., if the previous meaning representation applies a `Count` rule which returns a number, the next rule cannot be `LookupKey` since the expected argument is a database column instead of a number). We use additional constrains to guarantee that the anaphoric links are valid; we cache the n most recently generated meaning representations, their consequents and antecedents, and return types. For the next meaning representation, we sample terminal-level rules which generate co-referential variables and their values, by sampling one of the cases below:

1. the co-referential variable is `Coref`, and its value is one of the previously generated meaning representations whose denotation is type checked and has no consequents. This results in the Exploitation structure.
2. the co-referential variable is `Coref`, and its value is one of the previously generated meaning representations whose denotation is type checked and has other consequents. This results in the Exploration structure.
3. the co-referential variable is `Union_coref` or `Intersection_coref`, and its value is constructed from two previous meaning representations whose denotations are type checked. Moreover, the two meaning representations should not exhibit an Exploitation structure. This results in the Merging structure.
4. there is no co-referential variable in the meaning representation. This results in the Unrelated structure.

Step 2: Converting Meaning Representations to Templates For each meaning representation in the sequence, we retrieve the domain-general rule it uses, and look up the corresponding template. The template has missing entries, which are instantiated with domain-specific predicates, entities, or co-referential index.

Since meaning representations may be constructed with more than one domain-general rules, there may be more than one initial templates associated with each meaning representation. We combine these templates by merging their constrains. For example, the two templates ‘ $Result_1 = find [restaurants] \text{ with } [distance] < [500m]$ ’ and $Result_2 = find [Result_1] \text{ with } [price] > [50\$]$ are merged into $Result_1 = find [restaurants] \text{ with } [distance] < [500m] \text{ and } [price] > [50\$]$. This ensures that every meaning representation has exactly one canonical template representation.

Step 3: Annotating Utterances We display the sequence of instantiated templates to crowdworkers. Different from the single-turn case where crowdworkers summarize all the

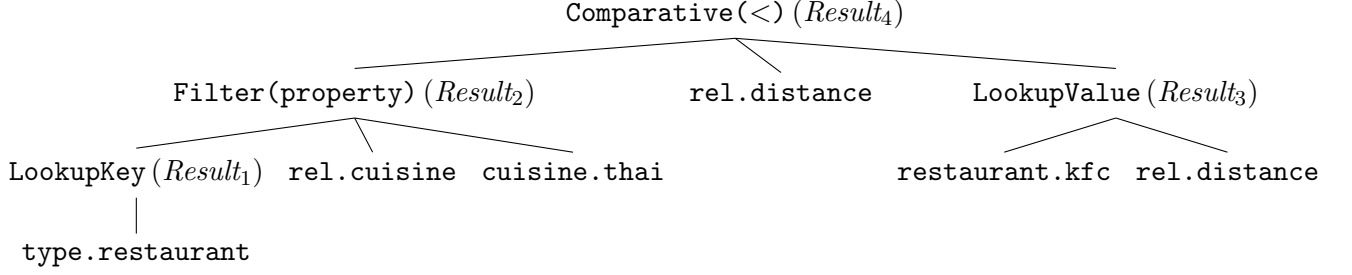


Figure 2: Derivation tree for meaning representation in Table 7. Domain-general rules are represented as non-terminal nodes, using the abbreviations shown in Table 2. For example, `LookupKey` refers to $\lambda s:(\text{lookupKey } (\text{var } s))$. Domain-specific and co-referential rules or variables are represent as terminal nodes.

templates into a single utterance, in the sequential case we ask them to paraphrase every template into an utterance, while expressing the discourse structure of the whole task (by using implicit or explicit co-reference). This results in a sequence of inter-related utterances paired with meaning representations.

4. Neural Semantic Parsing

A feature of our data elicitation method is that it explicitly models the compositional process underlying meaning representations, and caches the rules applied recursively to derive them. This characteristic allows us to train a neural semantic parser which generates meaning representations by predicting and composing rules therein. Figure 2 shows an example of the derivation our model predicts.

4.1 Transition Algorithm for Generating Derivations

We introduce an algorithm for generating tree-structured derivations with three classes of transition actions. The key insight underlying our algorithm is to define a canonical generation order, which produces a tree as a sequence of configuration-transition pairs $[(c_0, t_0), (c_1, t_1), \dots, (c_m, t_m)]$. In this work, we use top-down order for generation.

The top-down system is specified by tuple $c = (\sum, \pi, \sigma, N, P)$ where \sum is a stack used to store partially complete tree fragments, π denotes non-terminal rules to be generated in the derivation tree, σ denotes terminal rules to be generated, N is a stack of incomplete non-terminal rules, and P is a function indexing the position of a non-terminal pointer. The pointer indicates where subsequent rules should be attached (e.g., $P(X)$ means that the pointer is pointing to the non-terminal X and the next rule is generated underneath X). We take the initial configuration of the transition system to be $c_0 = ([], TOP, \varepsilon, [], \perp)$, where TOP stands for the root node of the tree, ε represents an empty string, and \perp represents an unspecified function. The top-down system employs three classes of transition operations defined in Table 9:

Top-down Transitions	
NT(X)	$([\sigma X'], X, \varepsilon, [\beta X'], P(X')) \Rightarrow ([\sigma X', X], \varepsilon, \varepsilon, [\beta X', X], P(X))$
TER(x)	$([\sigma X'], \varepsilon, x, [\beta X'], P(X')) \Rightarrow ([\sigma X', x], \varepsilon, \varepsilon, [\beta X', x], P(X'))$
RED	$([\sigma X', X, x], \varepsilon, \varepsilon, [\beta X', X], P(X)) \Rightarrow ([\sigma X', X(x)], \varepsilon, \varepsilon, [\beta X'], P(X'))$

Table 9: Transition actions for the top-down generation system. Stack \sum is represented as a list with its head to the right (with tail σ).

- NT(X) creates a new subtree rooted by a non-terminal rule X . The non-terminal X is pushed on top of the stack and written as $X($. The token $($ implies X is incomplete and subsequent rules are generated as children underneath X . In our semantic formalism, X can be any domain-general rule in Table 2.
- TER(x) creates a new terminal rule x . The terminal x is pushed on top of the stack, written as x . In our semantic formalism, x is one category of the domain-specific or co-referential rules shown in the first column of Table 3 and Table 4. At this stage, TER(x) simply generates a variable (i.e., a category) whose value is yet to be determined by a domain-specific classifier.
- RED is the reduce operation which indicates that the current subtree being generated is complete. The non-terminal rule of the current subtree will be applied to the children rules underneath (i.e. a beta reduction in lambda calculus), formulating a large piece of meaning representation. This large piece of meaning representation serves as a single child to its predecessor non-terminal, and subsequent children rules will be attached to the predecessor. Stack-wise, RED recursively pops children rules (which can be either terminal rules or partial meaning representations) on top until an incomplete non-terminal rule is encountered. The non-terminal is popped as well and beta-reduced, after which the larger piece of meaning representation is pushed back to the stack as a single closed constituent, written for example as $X1(X2, X3)$.

The space of transition actions is domain-general and language dependent. It covers operations that generate non-terminal-level rules (i.e., domain-general rules), terminal-level variables (for domain-specific and co-referential rules), and RED which enforces tree structures. Notice that TER(x) generates terminal-level variables only but not their values. This makes the entire generation algorithm portable across domains or transferable to new domains. Specific values for terminal-level variables (e.g., which domain-specific rule should be used, or which utterance the anaphora links to) will be predicted by subsequent classifiers, as an instance of hierarchical classification. The sequence of transition actions which generate the tree in Figure 2 is as follows:

```
NT(Comparative(<)), NT(Filter(property)), NT(LookupKey), TER(Entity),
RED, TER(Binary_predicate), TER(Entity), RED, TER(Binary_predicate), RED,
NT(LookupValue), TER(Entity), TER(Binary_predicate), RED, RED.
```

4.2 Neural Network Realizer

We model the above generation algorithm with a neural **sequence-to-tree** model, which encodes the utterance and then predicts a sequence of transition actions which generate rules to construct the derivation tree of the meaning representation.

Encoder Each utterance x is encoded with a bidirectional LSTM (Hochreiter & Schmidhuber, 1997). A bidirectional LSTM is comprised of a forward LSTM and a backward LSTM. The forward LSTM processes a variable-length sequence $x = (x_1, x_2, \dots, x_n)$ by incrementally adding new content into a single memory slot, with gates controlling the extent to which new content should be memorized, old content should be erased, and current content should be exposed. At time step t , the memory \vec{c}_t and the hidden state \vec{h}_t are updated with the following equations:

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \hat{c}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W \cdot [\vec{h}_{t-1}, x_t] \quad (1)$$

$$\vec{c}_t = f_t \odot \vec{c}_{t-1} + i_t \odot \hat{c}_t \quad (2)$$

$$\vec{h}_t = o_t \odot \tanh(\vec{c}_t) \quad (3)$$

where i , f , and o are gate activations; W denotes the weight matrix. For simplicity, we denote the recurrent computation of the forward LSTM as:

$$\vec{h}_t = \overrightarrow{\text{LSTM}}(x_t, \vec{h}_{t-1}) \quad (4)$$

After encoding, a list of token representations $[\vec{h}_1, \vec{h}_2, \dots, \vec{h}_n]$ within the forward context is obtained. Similarly, the backward LSTM computes a list of token representations $[\overleftarrow{h}_1, \overleftarrow{h}_2, \dots, \overleftarrow{h}_n]$ within the backward context as:

$$\overleftarrow{h}_t = \overleftarrow{\text{LSTM}}(x_t, \overleftarrow{h}_{t+1}) \quad (5)$$

Finally, each input token x_i is represented by the concatenation of its forward and backward LSTM state vectors, denoted by $h_i = \vec{h}_i : \overleftarrow{h}_i$. The list storing token vectors for the entire utterance x is denoted by $b = [h_1, \dots, h_k]$, where k is the length of the utterance.

Decoder After the utterance is encoded, the derivation tree of the corresponding meaning representation is generated with a stack-LSTM decoder, whose state updates depend on the class of transition operations. Specifically, transition actions **NT** and **TER** change the stack-LSTM representation s_t as in a vanilla LSTM:

$$s_t = \text{LSTM}(y_t, s_{t-1}) \quad (6)$$

where y_t denotes the newly generated non-terminal or terminal rule.

Transition action **RED** recursively pops the stack-LSTM states as well as corresponding tree nodes on the output stack. The popping stops when a state for a non-terminal rule

is reached and popped, after which the stack-LSTM reaches an intermediate state $s_{t-1:t}$.⁴ The representation of the completed subtree u , which represents a large piece of meaning representation, is then computed as:

$$u = W_u \cdot [p_u : c_u] \quad (7)$$

where p_u denotes the non-terminal rule in the subtree, c_u denotes the average of the children embeddings underneath, and W_u denotes the weight matrix. Note that c_u can also be computed with more advanced method such as a recurrent neural network (Dyer et al., 2016). Finally, the subtree embedding u serves as the input to the LSTM and updates $s_{t-1:t}$ to s_t as:

$$s_t = \text{LSTM}(u, s_{t-1:t}) \quad (8)$$

Predictions for Transition Actions At each time step t , the decoder needs to predict the next transition action. This prediction is conditioned on the utterance (represented by the bidirectional-LSTM states $b = [h_1, \dots, h_k]$), and the partially completed derivation tree (represented by the current state of the stack-LSTM s_t). To make the prediction, we compute an adaptive representation of the utterance \bar{b}_t with a soft attention mechanism:

$$u_t^i = V \tanh(W_b b_i + W_s s_t) \quad (9)$$

$$\alpha_t^i = \text{softmax}(u_t^i) \quad (10)$$

$$\bar{b}_t = \sum_i \alpha_t^i b_i \quad (11)$$

where W_b and W_s are weight matrices and V is a weight vector. We then combine \bar{b}_t and s_t with a feed-forward neural network (Equation (12)) to yield a feature vector for the generation system. Finally, softmax is taken to obtain the parameters of the multinomial distribution over actions:

$$a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f [\bar{b}_t, s_t])) \quad (12)$$

where W_{oa} and W_f are weight matrices. In greedy decoding, a_{t+1} is selected as the action which has the highest probability.

In cases where transition action $\text{TER}(\mathbf{x})$ generates variables of predicates, entities, or co-reference, we need to further predict their values.

Predictions for Variable Values For predicates and entities, we use a soft attention layer similar to Equations (9)–(11) to compute the adaptive utterance representation \bar{b}_t ; and then predict specific entities or predicates with another softmax classifier as:

$$y_{t+1} \sim \text{softmax}(W_{oy} \tanh(W_f [\bar{b}_t, s_t])) \quad (13)$$

which outputs multinomial distribution over predicates or entities, and this set of neural parameters is domain-specific.

4. We use $s_{t-1:t}$ to denote the interim state from time step $t-1$ to t , after popping; s_t denotes the final LSTM state after the subtree representation is pushed back to the stack (as explained in the following).

To predict the value of co-referential variables, we rely on a self-attention network (Cheng, Dong, & Lapata, 2016; Parikh, Täckström, Das, & Uszkoreit, 2016) to determine which utterances it co-refers with. The number of predictions is determined by the type of the co-referential variable. For variable **Coref**, we need to predict one anaphoric link, whereas for **Union_coref** and **Intersection_coref** we need to predict two.

Given utterance x , our previous bidirectional LSTM encoder obtains a list of forward representations $[\vec{h}_1, \dots, \vec{h}_n]$ and backward representations $[\overleftarrow{h}_n, \dots, \overleftarrow{h}_1]$. We use the concatenation of \vec{h}_n and \overleftarrow{h}_1 as utterance representation X , which is fed to the self-attention network. Given the current utterance embedding X_c and the previous utterance embeddings X_1, \dots, X_{c-1} , the self-attention network computes the relation distribution between X_c and every previous utterance in the list of $[X_1, \dots, X_{c-1}]$. This is accomplished by first computing score u_c^i for each previous utterance in the list, with index i ranging from 1 to $c-1$:

$$u_c^i = W_v \tanh(W_i X_i + W_X X_c) \quad (14)$$

Then a softmax classifier is used to obtain a_c^i , the probability that the current utterance X_c co-refers with i th utterance X_i :

$$a_c^i = \text{softmax}(u_c^i) \quad (15)$$

where the softmax is taken over all previous utterances in the list, and W s are weight parameters. We then select the most probable utterance as the value of **Coref**, or the top two most probable utterances as the value of **Union_coref** or **Intersection_coref**.

4.3 Context-dependent Parsing (Optional)

The neural semantic parser presented above is able to generate meaning representations with and without co-reference, and thus handle both single-turn and sequential utterances. However, it should be noted that when parsing sequential utterances, our semantic parser does not make an explicit use of context: the generation of a co-referential variable relies on the current utterance only, before the variable’s value is predicted within context. In the following, we extend our model to make an explicit use of context when parsing sequential utterances.

The central idea is to maintain a context vector for the current user session. This vector is used as an additional feature to the softmax classifier (Equation (12)) that predicts transition operations. To compute the context vector, we use the same self-attention network described in Equation (14). Given current utterance x_c with representation X_c , the network computes score u_c^i for each of the previous utterances x_i , with representation X_i and index i ranging from 1 to $c-1$:

$$u_c^i = W_v \tanh(W_i X_i + W_X X_c) \quad (16)$$

We then compute a context representation for the current utterance, denoted by \bar{X}_c :

$$a_c^i = \text{softmax}(u_c^i) \quad (17)$$

$$\bar{X}_c = \sum a_c^i * X_c \quad (18)$$

#rules	1				2				3				4				All			
Domain	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO	Q	Tp	Tk	WO
meeting	378	191	9.1	1.41	570	537	16.20	2.21	254	254	16.2	2.81	46	46	21.37	3.19	1,248	1,028	12.59	2.21
employees	322	240	8.96	1.34	320	319	13.47	2.95	486	486	18.2	3.66	268	268	22.37	3.12	1,396	1,313	15.79	3.12
hotel	170	146	8.99	1.91	358	542	16.86	3.64	542	542	16.86	4.11	433	433	19.89	4.05	1,503	1,479	15.87	4.05
restaurant	132	98	8.14	1.40	301	295	12.74	3.41	495	495	16.74	3.74	311	311	20.02	3.66	1,239	1,199	15.68	3.66
disease	283	212	9.3	1.29	301	455	14.23	3.52	455	455	19.49	5.65	213	213	23.95	4.48	1,252	1,176	16.68	4.48
medication	136	102	8.53	1.31	252	246	11.89	2.35	435	435	16.09	3.17	247	247	20.04	2.91	1,070	1,030	15.05	2.91

Table 10: SINGLETURN dataset: number of utterances (Q), number of templates (Tp), average number of tokens (Tk) and token overlap between queries and templates (WO) per domain and overall (All).

The context representation \bar{X}_c is used at every time step when the decoder generates the derivation tree of x_c , by extending Equation (12) as:

$$a_{t+1} \sim \text{softmax}(W_{oa} \tanh(W_f[\bar{b}_t, s_t, \bar{X}_c])) \quad (19)$$

We will experimentally verify whether modeling context explicitly is useful for parsing sequential utterances.

5. Experiments

In this section, we describe a range of experiments conducted to evaluate the framework which builds neural semantic parsers from domain ontologies. Specifically, we will present experimental results and analysis for both single-turn and sequential utterances.

5.1 Data Elicitation

5.1.1 SINGLE-TURN UTTERANCES

We elicited single-turn utterances for six domains simulating database querying tasks. Two domains relate to company management (meeting and employees databases), two concern recommendation engines (hotel and restaurant databases), and two target healthcare applications (disease and medication databases).

The dataset was collected with Amazon Mechanical Turk (AMT). Across domains, the total number of querying tasks (described by templates) that the framework generated was 7,225. These tasks were sampled randomly without replacement to show to annotators. Each annotator saw three tasks per HIT and was paid 0.3\$. After removing repeated query-meaning representation pairs, we collected a semantic parsing dataset of 7,708 examples. The average amount of time annotators spent on each domain was three hours. We evaluated the correctness of 100 randomly chosen utterances, and the accuracy was 81%. For the failed examples, annotators generated utterances constitute partial or wrong descriptions of the meaning representations.

Table 10 shows various statistics of the elicited dataset which we call SINGLETURN. These include the number of utterances (Q) we obtained for each domain broken down

according to the depth of compositionality, the number of templates (Tp) per domain, the average number of tokens (Tk) per utterance in each domain, and the token overlap (Ov) between the utterances and the corresponding templates. The number of utterances collected at each compositional level is dependent on the space of predicates in each domain. We see that utterance length does not vary drastically among domains even though the average number of tokens is affected by the verbosity of entities and predicates in each domain. We use word overlap as a measure of the amount of paraphrasing. We see that utterances in the disease domain deviate least from their corresponding templates while utterances in the meeting domain deviate most. This number is affected by the degree of expert knowledge required for annotation in each domain.

Table 11 presents examples of the utterances we elicited for the six domains together with their corresponding templates.

5.1.2 COMPARISON TO WANG ET AL. (2015)

Our data collection method is closely related to Wang et al. (2015) in that we also ask annotators to paraphrase artificial expressions into natural sounding ones. In their approach, annotators paraphrase a single artificial description. In comparison, we aim to handle more complex tasks involving compositional intentions. For such tasks it is not easy to merge all meanings into a single formal description. We therefore represent the meanings with a sequence of templates, and ask annotators to summarize the templates into a natural language utterance.

We directly evaluated how annotators perceive our templates compared to single descriptions, with respect to the compositionality of meaning representations. For each domain we randomly sampled 24 meaning representations described by templates (144 tasks in total) and derived the corresponding artificial description using Wang et al.’s (2015) grammar. The output from both approaches was also paired with a task description (manually created by us) which explained the task (see Table 14 for examples). Annotators were asked to rate how well the single artificial sentence and the templates corresponded to the natural language description according to two criteria: (a) intelligibility (how easy is the artificial language to understand?) and (b) accuracy (does it match the intention of the task?). Participants used a 1–5 rating scale where 1 is worst and 5 is best. We elicited 5 responses per task.

Table 12 summarizes the mean ratings for each domain and overall. As can be seen, our approach generally receives higher ratings for Intelligibility and Accuracy. For both types of ratings, templates significantly outperform the individual descriptions for all domains but meetings. Table 13 shows a breakdown of results according to the depth of compositionality. Utterances of compositional depth 1 and 2 can be easily described by one sentence, and our template-based approach has no clear advantage over Wang et al. (2015). However, when the compositional depth increases to 3 and 4, templates are perceived as more intelligible and accurate across domains; all means differences for depths 3 and 4 are statistically significant ($p < 0.01$). Further qualitative analysis suggests that our approach receives higher ratings in cases where the output of the grammar from Wang et al. (2015) involves various propositional attachment ambiguities. The ambiguities are common when the compositional depth increases (Example 1 in Table 14), when the query contains con-

Domain	Templates	Query
meeting	$R_1 = \text{find } [\text{location}] \text{ of } [\text{annual review}]$	<i>what location will the annual review take place</i>
meeting	$R_1 = \text{find } [\text{location}] \text{ of } [\text{annual review}]$ $R_2 = \text{find all } [\text{meetings}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{location}] \text{ is not } [R_1]$ $R_4 = \text{find } [R_3] \text{ with smallest number of } [\text{attendee}]$	<i>which meeting is not held in the same venue as annual review, and attracts the least amount of attendance</i>
employees	$R_1 = \text{find all } [\text{employees}]$ $R_2 = \text{find } [R_1] \text{ with smallest number of } [\text{projects}]$	<i>which employee is assigned minimum projects</i>
employees	$R_1 = \text{find all } [\text{employees}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{salary}] < [5000]$ $R_3 = \text{find } [R_1] \text{ with } [\text{salary}] > [15000]$ $R_4 = \text{find } [R_2 \text{ or } R_3] \text{ where } [\text{division}] \text{ is } [\text{IT}]$	<i>for those employees in the IT division, who are paid less than 5000 or more than 15000</i>
hotel	$R_1 = \text{find all } [\text{hotels}]$ $R_2 = \text{find } [R_1] \text{ where } [\text{distance}] \text{ is } [300]$ $R_3 = \text{find } [R_2] \text{ which satisfies } [\text{free cancellation}]$	<i>which hotel is at 300 metres and doesn't charge a cancellation fee</i>
hotel	$R_1 = \text{find all } [\text{hotels}]$ $R_2 = \text{find } [R_1] \text{ where } [\text{location}] \text{ is } [\text{oxford street}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{room type}] \text{ is } [\text{single or double}]$ $R_4 = \text{find } [R_3] \text{ which satisfies } [\text{has free wifi}]$	<i>which hotel in oxford has single or double room? the hotel should has free wifi too</i>
restaurant	$R_1 = \text{find all } [\text{restaurants}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{number of reviews}] > [100]$ $R_3 = \text{count elements in } [R_2]$	<i>how many restaurants have more than 100 reviews</i>
restaurant	$R_1 = \text{find all } [\text{restaurants}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{number of reviews}] > [500]$ $R_3 = \text{find } [R_2] \text{ with largest number of } [\text{cuisine}]$ $R_4 = \text{find } [R_3] \text{ which satisfies } [\text{has outdoor seatings}]$	<i>for the restaurants with more than 500 reviews, look for those with the largest variety of food, and then those with seats outside</i>
disease	$R_1 = \text{find all } [\text{diseases}]$ $R_2 = \text{find } [R_1] \text{ with smallest } [\text{incubation period}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{symptom}] \text{ is not } [\text{bleeding}]$ $R_4 = \text{count elements in } [R_3]$	<i>how many diseases having the smallest incubation period don't result in bleeding</i>
disease	$R_1 = \text{find } [\text{symptom}] \text{ of } [\text{fever}]$ $R_2 = \text{find all } [\text{diseases}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{symptom}] \text{ is } [R_1]$ $R_4 = \text{find } [R_3] \text{ with largest } [\text{incubation period}]$	<i>which disease has the same symptom as fever, and has the longest incubation period</i>
medication	$R_1 = \text{find all } [\text{medications}]$ $R_2 = \text{find } [R_1] \text{ which satisfies } [\text{for adult only}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{target symptom}] \text{ is } [\text{bleeding}]$ $R_4 = \text{find } [R_3] \text{ where } [\text{side effect}] \text{ is } [\text{headache}]$	<i>what adult medications treat bleeding in exchange for a headache</i>
medication	$R_1 = \text{find all } [\text{medications}]$ $R_2 = \text{find } [R_1] \text{ where } [\text{target symptom}] \text{ is } [\text{headache}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{category}] \text{ is } [\text{physician or pharmacist}]$ $R_4 = \text{find } [R_3] \text{ which satisfies } [\text{requires prescription}]$	<i>find the medicine for headache, with a category of physician or pharmacist; and the medicine requires prescription</i>

Table 11: Examples of templates (filled values shown within brackets), and elicited utterances across six domains. R is a shorthand for *Result*.

Domain	Intelligibility		Accuracy		Combined	
	S	T	S	T	S	T
meeting	3.54	3.81	3.86	4.02	3.70	3.91
employee	<u>3.58</u>	<u>4.13</u>	<u>3.48</u>	<u>4.43</u>	<u>3.53</u>	<u>4.28</u>
hotel	3.81	3.96	<u>3.79</u>	<u>4.29</u>	<u>4.12</u>	<u>3.80</u>
restaurant	<u>3.93</u>	<u>4.23</u>	3.80	3.75	<u>3.86</u>	<u>4.13</u>
disease	3.41	3.69	<u>3.68</u>	<u>4.02</u>	<u>3.55</u>	<u>3.86</u>
medication	3.83	3.97	<u>3.83</u>	<u>4.40</u>	<u>3.83</u>	<u>4.19</u>
All	3.96	3.67	<u>3.55</u>	<u>4.03</u>	<u>3.70</u>	<u>4.06</u>

Table 12: Comparison between artificial descriptions (S; Wang et al., 2015) and our template-based approach (T). Mean ratings are shown per domain and overall. Combined is the average of Intelligibility and Accuracy. Means are underlined if their difference is statistically significant at $p < 0.05$ using a post-hoc Turk test.

Depth	Intelligibility		Accuracy		Combined	
	S	T	S	T	S	T
1	4.00	4.29	4.05	4.29	4.03	4.26
2	4.11	4.18	4.03	4.18	4.07	4.02
3	<u>3.61</u>	<u>4.09</u>	<u>3.60</u>	<u>4.01</u>	<u>3.58</u>	<u>4.01</u>
4	<u>3.56</u>	<u>4.28</u>	<u>3.35</u>	<u>4.25</u>	<u>3.60</u>	<u>4.10</u>

Table 13: Comparison between artificial sentences (S; Wang et al., 2015) and template-based approach (T) for varying compositionality depths. Mean ratings are aggregated across domains. Combined is the average of Intelligibility and Accuracy. Means are underlined if their difference is statistically significant at $p < 0.01$ using a post-hoc Turk test.

junction and disjunction (Example 2 in Table 14), and when a sub-query acts as object of comparison in a longer query (Example 3 in Table 14).

5.1.3 SEQUENTIAL UTTERANCES

We also used AMT to elicit sequential utterances for two domains, namely restaurant and hotel domains. We simulated a database querying task, where the user asks a series of questions in order to accomplish an information need. Each user session (i.e., querying task) included a maximum amount of five co-reference, covering the four co-referential structures described in Section 3.4. The Exploitation structure is the building block for all sessions, since it is the most natural, the user asks follow-up queries to previous ones. Each session includes an additional structure randomly selected from Exploration, Merging, and Unrelated, which respectively simulate comparisons between choices, logical union and intersection, and topic shift. In our experiment, these three structures can co-exist with Exploitation, but not with each other. More complex co-referential structures (e.g., two explorations in the same session) are not common in practice, although they can be constructed from the four basic structures.

Task description	Our templates	Wang et al. (2015)
We have a database of diseases and would like to find diseases which have fever as their symptom. These diseases should be treatable with antibiotics. Their incubation period is longer than a day. If you have such a disease you should see a doctor.	<i>R₁ = find the diseases whose symptom is fever</i> <i>R₂ = find R₁ whose treatment is antibiotics</i> <i>R₃ = find R₂ whose incubation period is longer than a day</i> <i>QR = find R₃ which require to see a doctor</i>	diseases whose symptom is fever whose treatment is aspirin whose incubation period is larger than a day which require to see a doctor
We have a database of diseases and would like to find diseases which have fever as their symptom; amongst them, we would like to find those with heart disease as complication. Finally, we want to find all diseases that can be treated with antibiotics.	<i>R₁ = find the diseases whose symptom is fever</i> <i>R₂ = find the diseases whose complication is heart disease</i> <i>QR = find R₁ and R₂ whose treatment is antibiotics</i>	disease whose symptom is fever and disease whose complication is heart disease whose treatment is antibiotics
We have a database of diseases. We would like to first find the incubation period of fever; and then find the diseases which have incubation period longer than fever; these diseases can be also treated with antibiotics.	<i>R₁ = find incubation period of fever</i> <i>R₂ = find diseases whose incubation period is larger than R₁</i> <i>QR = find R₂ whose treatment is antibiotics</i>	diseases whose incubation period is larger than incubation period of fever whose treatment is antibiotics

Table 14: Templates and artificial sentences (Wang et al., 2015) shown to AMT crowdworkers together with task description. Examples are taken from the disease domain. *R* and *QR* are shorthands for *Result* and *Query Result*, respectively.

	hotel	restaurant
Input vocabulary (for natural language) size	3,813	4,628
Output vocabulary (for database) size	386	386
average #utterances per session	3.98	4.01
average #tokens per utterance	9.30	9.11
average #word overlap per utterance-template	4.22	4.07

Table 15: Statistics of the SEQUENTIAL dataset.

For each domain, we generated meaning representations corresponding to four types of sessions consisting of Exploitation (only), Exploration, Merging, and Unrelated. The number of sessions (described by templates) for each type was 3,000. Sessions were sampled randomly without replacement to show to annotators. Each annotator saw two tasks per HIT and was paid 0.2\$. The average amount of time annotators spent on each domain was three hours. Finally, we obtained 12,000 examples for each domain. Table 15 shows basic statistics of our dataset which we call SEQUENTIAL. These include input and output vocabulary size, the number of utterances per user session, the number of tokens per utterance, and the word overlap between an utterance and its corresponding template. As can be seen, SEQUENTIAL exhibits a significant amount of paraphrasing (more than half of the tokens in each template are paraphrased).

The dataset inevitably contains a certain amount of noise, including spelling mistakes, inaccurate paraphrasing, and wrong co-reference. However, we did not perform any manual postprocessing as our goal in this work is to simulate a real-world setting where the semantic parser has to learn under noise. Nevertheless, we evaluated 100 randomly selected sessions

(consisting of 531 queries in total). Amongst these, 79 sessions are correct, while the remaining 21 contain wrongly paraphrased utterances. The paraphrasing accuracy for all the 531 utterances is 94.4%. Aside from minor spelling errors, we found that all mistakes workers made related to co-reference. Most commonly, they ignored co-reference during paraphrasing. For example, instead of asking *of these restaurants, which ones have thai food*, an annotator may simply write *which restaurants have thai food*.

Table 16 and 17 present examples of the sequential utterances we elicited for the hotel and restaurant domains together with their co-reference structures and corresponding templates.

Domain	Structure	Templates	Queries
hotel	Exploitation	$R_1 = \text{find } [\text{hotels}] \text{ with } [\text{price rating}] \leq [\$\$]$ $R_2 = \text{find } [R_1] \text{ with number of } [\text{room type}] \geq [4]$ $R_3 = \text{find } [R_2] \text{ which satisfies } [\text{near the sea}]$ $R_3 = \text{find } [R_2] \text{ which satisfies } [\text{can be reserved}]$	<i>Which hotels have a price rating of 2 dollar signs or less?</i> <i>Can you find which of these have more than 4 different types of rooms?</i> <i>Among these, which are located near the sea?</i> <i>Among these, which takes reservations?</i>
hotel	Exploration	$R_1 = \text{find } [\text{hotels}] \text{ with largest number of } [\text{customer reviews}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{customer rating}] \geq [5 \text{ stars}]$ $R_3 = \text{find } [R_1] \text{ with } [\text{customer rating}] \geq [3 \text{ stars}]$	<i>Show me hotels with the largest number of customer reviews</i> <i>Which of those hotels have received a customer rating of 5 or more stars?</i> <i>Of the hotels with the largest number of reviews, which ones have a customer rating of 3 or more stars?</i>
hotel	Merging	$R_1 = \text{find } [\text{hotels}] \text{ with largest number of } [\text{room type}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{customer rating}] \geq [5 \text{ stars}]$ $R_3 = \text{find } [R_1] \text{ with } [\text{distance}] \leq [500\text{m}]$ $R_4 = \text{find } [R_2 \text{ and } R_3] \text{ which satisfies } [\text{car parks}]$	<i>Which hotel has the most variety of rooms?</i> <i>Which of these are rated at 5 stars or more?</i> <i>Which of the previous hotels are within 500 meters to me?</i> <i>Of all these hotels with 5 stars or near me, which offers car parks?</i>
hotel	Unrelated	$R_1 = \text{find } [\text{hotels}] \text{ with largest number of } [\text{room type}]$ $R_2 = \text{find } [R_1] \text{ with smallest } [\text{price rating}]$ $R_3 = \text{find } [\text{hotels}] \text{ which satisfies } [\text{airport shuttle}] \text{ and } [\text{private bathroom}]$	<i>What hotels have the largest amount of room types?</i> <i>Among those, which have the smallest price rating?</i> <i>Which hotels have an airport shuttle and private bathroom?</i>

Table 16: Examples of co-reference structures, templates (filled values shown within brackets) and elicited utterances for the hotel domain. R is a shorthand notation for *Result*.

5.1.4 CAN ANNOTATORS CREATE THEIR OWN TASKS?

In the data collection methods described so far, annotators are shown templates and asked to write down natural language expressions summarizing them. The fact that templates are generated automatically allows us to explore a potentially very large space of underlying meaning representation exhibiting compositionality and wide coverage. However, in practice, companies sometimes hire annotators who are in charge of creating domain-specific tasks on their own. These annotators are often trained with domain knowledge but lack in programming knowledge. We would like to further explore if annotators are able to create their own querying tasks and annotations, with the aid of templates.

Domain	Structure	Templates	Queries
restaurant	Exploitation	$R_1 = \text{find } [\text{restaurants}] \text{ with smallest } [\text{price rating}]$ $R_2 = \text{find } [R_1] \text{ with largest number of } [\text{food type}]$ $R_3 = \text{find } [R_2] \text{ which satisfies } [\text{good for groups}] \text{ and } [\text{has waiter service}]$ $R_4 = \text{find } [R_3] \text{ with } [\text{distance}] \leq [500\text{m}]$	<i>Show me the cheapest restaurants.</i> <i>Which of these have the largest variety of food?</i> <i>Of these, are there any restaurants that are good for groups and offer waiter service?</i> <i>Are any of those restaurants within half a kilometer to me?</i>
restaurant	Exploration	$R_1 = \text{find } [\text{restaurants}] \text{ with largest } [\text{customer rating}]$ $R_2 = \text{find } [R_1] \text{ with } [\text{distance}] \leq [500\text{m}]$ $R_3 = \text{find } [R_2] \text{ where } [\text{food type}] \text{ is } [\text{thai food}]$ $R_4 = \text{find } [R_2] \text{ where } [\text{food type}] \text{ is } [\text{american food}]$	<i>Which restaurants have the largest customer ratings?</i> <i>Which of these restaurant is within 500 meters?</i> <i>Show me those serves Thai food?</i> <i>Of the previous restaurants, find the ones which serves American food instead.</i>
restaurant	Merging	$R_1 = \text{find } [\text{restaurants}] \text{ where } [\text{location}] \text{ is } [\text{downtown}]$ $R_2 = \text{find } [R_1] \text{ which satisfies } [\text{has breakfast}]$ $R_3 = \text{find } [R_1] \text{ which satisfies } [\text{has lunch}]$ $R_4 = \text{find } [R_2 \text{ and } R_3] \text{ with largest number of } [\text{customer reviews}]$	<i>Show me restaurants located downtown?</i> <i>Which of these restaurants serve breakfast?</i> <i>Which restaurants serve lunch?</i> <i>Of all these restaurants with breakfast or lunch, which have the most customer reviews?</i>
restaurant	Unrelated	$R_1 = \text{find } [\text{restaurants}] \text{ which satisfies } [\text{can be reserved}]$ $R_2 = \text{find } [R_1] \text{ which satisfies } [\text{take credit card}]$ $R_3 = \text{find } [R_2] \text{ with number of } [\text{cuisine}] > [1]$ $R_4 = \text{find } [\text{restaurants}] \text{ with smallest } [\text{price rating}]$ $R_5 = \text{find } [R_4] \text{ with largest } [\text{customer rating}]$	<i>which restaurants have reservations option?</i> <i>which among these restaurants accept credit cards?</i> <i>among these restaurants find those with multiple types of cuisine?</i> <i>which restaurants have the smallest price rating?</i> <i>which among these restaurants have the largest customer rating?</i>

Table 17: Examples of co-reference structures, templates (filled values shown within brackets) and elicited utterances for the restaurant domain. R is a shorthand notation for *Result*.

Since our data elicitation method maps each domain-general rule into a template, which is both human-readable and machine interpretable, we can give annotators the freedom to manipulate templates for generating their own tasks. In order to specify a task, annotators need to selectively use a sequence of templates and instantiate them with domain-specific information. While they do so with an annotation tool at the front end, meaning representations are created automatically at the back end.

We conducted an experiment to explore this idea. Because it is not realistic to recruit a large number of domain experts to participate in our evaluation, we ran this experiment on AMT with annotators who were paid more to compensate for the increased workload. The experiment was conducted on the same six domains (meeting, employees, hotel, restaurant, disease and medication) used to elicit the SINGLETURN dataset. For each domain, annotators were given unfilled templates and a table describing naturalized predicates and entities of that domain. They were also given instructions and examples explaining how to use the templates and domain information. Annotators were then asked to chose the templates, fill them, and write down an utterance summarizing the task. We recruited 50 workers for each domain (300 in total) and each was asked to complete two tasks in one HIT worth 1\$.

Table 18 shows the statistics of the data we obtained. Workers tend to use more than three templates (on average) per task, and the degree of paraphrasing is increased compared

Domain	Tk	WO	Tp	Acc
meeting	16.71	2.83	3.46	0.925
employees	18.64	3.71	3.33	0.938
hotel	16.97	4.08	3.37	0.969
restaurant	17.33	3.72	3.36	0.943
disease	18.95	5.12	3.12	0.925
medication	17.25	3.26	3.15	0.989

Table 18: Average number of tokens (Tk), token overlap between queries and templates (WO), average number of templates (Tp) and proportion of parsable templates (Acc) per domain when annotators are given flexibility to generate querying tasks.

to the original data collection method where tasks are generated automatically. Moreover, across domains, 90% of the meaning representations created by AMT annotators are executable. Although we envisage domain experts as the main users of such an annotation tool, the result indicates that (with proper instructions and examples as well as adequate pay) regular AMT annotators can also generate meaningful meaning representations to some extent. Inspection of the output failures revealed two common reasons. Firstly, annotators filled in templates with wrong types. For example, one worker filled the `LookupKey` template with an entity (e.g., *find all [annual review]*) instead of a database key; and another worker used `Count` as the first template followed by `Filter`, however type constraints require `Filter` to take a set of entities as argument instead of a number (as returned by `Count`). Secondly, annotators ignored information in the table and created tasks using non-existing domain information. Since meaning representations can be type-checked for validity automatically, providing feedback on the fly may yield a higher percentage of executable meaning representations.

5.2 Semantic Parsing

In this section, we evaluate our neural semantic parser (Section 4) on the two datasets we collected (SINGLETURN and SEQUENTIAL).

5.2.1 SEMANTIC PARSING FOR SINGLE-TURN UTTERANCES

In the single-turn experiments, our parser parses an utterance into a meaning representation which does not contain co-referential variables⁵.

Our parser was trained on the SINGLETURN dataset presented in Section 5.1. Training/validation/testing splits are 0.7, 0.1 and 0.2. All LSTMs had one layer with 150 dimensions. The word embedding size and rule embedding size were set to 50. A dropout of 0.5 was used on the input features of the softmax classifiers in Equations (13)–(15) (and (16)–(19)). Momentum SGD was used to update the parameters of the model. We compared our model with two baselines, a sequence-to-sequence (S2S) parser (Dong & Lapata, 2016; Jia & Liang, 2016) which converts an utterance to a meaning representation in string

5. During the construction of the meaning representation, co-referential variables are always replaced with their values—which are antecedent meaning representations.

Model	meeting		employees		hotel		restaurant		disease		medication	
	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM
S2S	37.2	43.2	14.3	17.5	24.5	31.2	21.3	29.0	16.7	23.2	15.5	16.7
S2T	41.2	46.8	21.4	28.2	31.5	43.5	25.4	35.9	22.4	34.4	28.2	33.9
S2D	45.6	54.0	27.8	35.5	39.2	52.6	47.2	49.5	26.9	44.6	35.8	46.2

Table 19: Performance on various domains (test set) using exact match (ExM) and semantic match (SeM).

format, and a sequence-to-tree (S2T) parser which converts an utterance directly to an output tree (Cheng et al., 2017b). As an example, S2S generates the meaning representation (`filter (Result1) (rel.distance) < (num 500)`), S2S from left to right including auxiliary brackets (and). S2T generates the same meaning representation as a tree using cues from brackets: `Result1`, `rel.distance`, `<`, and `(num 500)` are all attached as children nodes to non-terminal `filter`. Note that S2T is a reasonable model for recursive meaning representations, whose structure reveals how meaning representations are compositionally obtained. However, the generation process of S2T is not interpretable for non-recursive meaning representations. Different from S2T, our model (denoted by S2D) generates the representation for the same example with a non-terminal rule `filter(<)`, and three terminal rules `Coref`, `BinaryPredicate` and `Entity`. These rules are then composed to derive the final logical form.

Table 19 shows results on the test set of each domain using exact match as the evaluation metric (ExM). Our sequence-to-derivation model (S2D) yields substantial gains over the baselines (S2S and S2T) across domains. However, a limitation of exact match is that different meaning representations may be equivalent due to the commutativity and associativity of rule applications. They can eventually execute to the same denotation. For example, two subsequent `Filter` rules in a meaning representation are interchangeable. For this reason, we additionally compute the number of meaning representations that match the gold standard at the semantic or denotation level. Again, we find that the S2D outperforms related baselines by a wide margin. This result is not surprising, since S2D explicitly models the compositionality of meaning representations.

We conducted further experiments by training a single model on data from all domains, and testing it on the test set of each individual domain. As the results in Table 20 reveal, we obtain gains for most domains on both metrics of exact and semantic match. Our results agree with previous work (Herzig & Berant, 2017) which improves semantic parsing accuracy by training a single sequence-to-sequence model over multiple domains. Since domain-general aspects are shared in our model, cross-domain training offers our parser more supervision cues.

5.2.2 SEMANTIC PARSING FOR SEQUENTIAL UTTERANCES

Next, we evaluate our neural semantic parser on sequential utterances. Specifically, each utterance in a user session is paired with a meaning representation which can contain co-referential variables. The goal of the parser is to parse every utterance in the session.

Model	ExM	SeM
meeting	(45.6) 48.8	(54.0) 56.8
employees	(27.8) 31.7	(35.5) 41.4
hotel	(39.2) 41.8	(52.6) 56.1
restaurant	(47.2) 33.1	(49.5) 48.8
disease	(26.9) 30.7	(44.6) 48.3
medication	(35.8) 37.4	(46.2) 51.8

Table 20: Sequence-to-derivation tree model (S2D) trained on all six SINGLETURN domains and evaluated on the test set of each domain. Results of S2D when trained and tested on a single domain are shown within brackets.

	Model	Exploitation		Exploration		Merging		Unrelated		All	
		ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM	ExM	SeM
restaurant	S2S	46.9	23.8	46.2	23.2	38.3	3.10	38.3	3.80	42.0	44.5
	S2T	69.2	40.7	64.1	45.5	50.9	4.70	50.4	6.30	57.9	61.2
	S2D	72.5	42.5	67.7	47.1	53.1	5.70	54.3	7.00	60.2	25.1
	S2D _C	73.6	42.8	68.1	47.1	54.6	5.80	54.9	7.1	62.4	25.1
hotel	S2S	48.3	24.6	48.2	19.5	43.6	6.3	41.5	4.10	44.5	13.6
	S2T	66.9	42.1	72.3	54.7	62.4	9.6	53.8	5.40	61.2	25.2
	S2D	73.8	43.7	77.8	56.8	65.4	11.8	57.2	6.70	66.9	29.7
	S2D _C	73.7	43.6	78.8	56.8	65.0	11.8	59.5	7.10	67.0	29.8

Table 21: Sequential semantic parsing results on the restaurant and hotel domains.

Our semantic parser was trained and test on the SEQUENTIAL dataset introduced in Section 3.4. We adopt the same experimental setup as in our single-turn experiments. Training/validation/testing splits are 0.7, 0.1 and 0.2. All LSTMs have one layer with 150 dimensions, and all word and rule embeddings have size 50. Recall that we use a self-attention network to compute the value of co-referential variables. This network relies on a bidirectional-LSTM to compute utterance vectors. In our experiments, the weights of this bidirectional-LSTM are shared with the encoder of the semantic parser. The encoder bidirectional-LSTM computes a list of token vectors, instead of a single vector per utterance. We compare this parser (S2D) against a context dependent parser (S2D_C) and two baselines, a vanilla sequence-to-sequence models (S2S) and sequence-to-tree (S2T) model.

For evaluation, we primarily consider the parsing accuracy of each individual utterance (ExM). This accuracy is first averaged within each user session and then averaged over all sessions. Similar to the single-turn evaluation, we additionally measure the match at the semantic or denotation level (SeM). We also break down our results based on data sessions involving different co-referential structures: with Exploitation only, and Exploitation combined with Exploration/Merging/Unrelated.

Table 21 shows our semantic parsing results, which clearly reveal the advantage of S2D over S2S and S2T. We see that results for semantic match are rather low, indicating that it is

challenging to parse all utterances in a session correctly. Comparing among various sessions, those with Merging and Unrelated result in lowest accuracy. To better understand how the model does on Exploration, Merging and Unrelated, we evaluate parsing performance (precision, recall, and F1) for specific utterances involving these structures.

For the Exploration structure, precision, recall and F1 are 84.4%, 73.6%, 78.6% respectively for the restaurant domain (87.7%, 71.9%, 79.0% for the hotel domain). For the Merging structure (which involves union or intersection), precision, recall, and F1 are 88.6%, 21.4%, and 34.5%, respectively for the restaurant domain (94.8%, 43.2%, and 59.4% for the hotel domain). The low recall indicates that most of the union or intersection symbols in the meaning representations are not discovered correctly. Finally, in the Unrelated structure, we have two independent querying tasks in one session. We predict how the model performs on predicting the first utterance of the second querying task, which indicates the model’s ability to detect topic shift. Precision, recall, and F1 are 81.9%, 63.8%, and 71.7%, respectively for the restaurant domain (82.8%, 68.1% and 74.7% for the hotel domain). Overall, we see that the parser does fairly well on Exploitation, Exploration, and Unrelated, while it fails to recognize most Merging structures.

Table 21 also shows the results of the context-dependent semantic parser (S2D_C). Across different co-reference structures, modeling context explicitly during decoding leads to some performance improvements; however, the gains are relatively small. We think this is due to the fact that a single utterance often provides enough cues to infer an explicit or implicit co-referential variable. Explicit co-reference is evidenced by pronouns (e.g., “*find restaurants in the oxford street?*” followed by “*those with car parks?*”), while in utterances with implicit co-reference the querying object is often missing (e.g., “*find restaurants in the oxford street?*” followed by “*with car parks?*”). It is not difficult for a non-context dependent neural network to learn these surface cues from training data. An exception is when an utterance contains nominal anaphora (e.g., “*find restaurants in the oxford street?*” followed by “*the oxford restaurants with car parks?*”). In this case, modeling context helps the parser identify a co-referential relation in the consequent utterance, where “*oxford*” refers to “*oxford street*” and “*the oxford restaurants*” refers to the denotation of the antecedent utterance.

6. Conclusions

In this work we examined a framework of building a neural semantic parser from a domain ontology. We focus on compositional tasks that represent complex human intentions. First, our data elicitation method starts from the formal language space, which computers are able to explore to generate meaning representations. These meaning representations are mapped to templates which are finally converted by humans into single or sequential utterances. Next, our neural semantic parser leverages the annotations we obtain to generate derivation trees of meaning representations, by explicitly modeling the composition of rules. Experimental results show that our framework provides an end-to-end solution for building a neural semantic parser from a domain ontology. This is useful when a new domain is introduced or a domain ontology is updated.

Future work will focus on devising smarter data generation methods with less human involvement. For example, natural language generation from formal descriptions may be

automated with a statistical generative model. Besides, we would like to apply similar data elicitation-modeling ideas to build conversational assistant (Liang, Klein, Gillick, Cohen, Arsenault, Clausman, Pauls, & Hall, 2018), which involves dialog flow management and decision making for optimum response strategies.

Bibliography

- Andreas, J., Vlachos, A., & Clark, S. (2013). Semantic Parsing as Machine Translation. In *Proceedings of Association for Computational Linguistics*, pp. 47–52.
- Artzi, Y., Das, D., & Petrov, S. (2014). Learning compact lexicons for ccg semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1273–1283.
- Artzi, Y., & Zettlemoyer, L. (2013a). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1(1), 49–62.
- Artzi, Y., & Zettlemoyer, L. (2013b). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1, 49–62.
- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *Proceedings of ICLR 2015*, San Diego, California.
- Berant, J., Chou, A., Frostig, R., & Liang, P. (2013a). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington, USA.
- Berant, J., Chou, A., Frostig, R., & Liang, P. (2013b). Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1533–1544, Seattle, Washington.
- Berant, J., & Liang, P. (2014). Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1415–1425, Baltimore, Maryland.
- Berant, J., & Liang, P. (2015). Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3, 545–558.
- Cai, J., Shin, R., & Song, D. (2017). Making neural programming architectures generalize via recursion..
- Chen, B., Sun, L., & Han, X. (2018). Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1, pp. 766–777.
- Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, Austin, Texas.
- Cheng, J., Reddy, S., Saraswat, V., & Lapata, M. (2017a). Learning structured natural language representations for semantic parsing. In *Proceedings of the 55th Annual*

- Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44–55, Vancouver, Canada.
- Cheng, J., Reddy, S., Saraswat, V., & Lapata, M. (2017b). Learning structured natural language representations for semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 44–55.
- Dong, L., & Lapata, M. (2016). Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33–43, Berlin, Germany.
- Dong, L., & Lapata, M. (2018). Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 731–742.
- Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 199–209, San Diego, California.
- Fan, X., Monti, E., Mathias, L., & Dreyer, M. (2017). Transfer learning for neural semantic parsing. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pp. 48–56.
- Gardner, M., & Krishnamurthy, J. (2017). Open-vocabulary semantic parsing with both distributional statistics and formal knowledge...
- Ge, R., & Mooney, R. J. (2005). A statistical semantic parser that integrates syntax and semantics. In *Proceedings of the ninth conference on computational natural language learning*, pp. 9–16. Association for Computational Linguistics.
- Gupta, S., Shah, R., Mohit, M., Kumar, A., & Lewis, M. (2018). Semantic parsing for task oriented dialog using hierarchical representations. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2787–2792.
- Herzig, J., & Berant, J. (2017). Neural semantic parsing over multiple knowledge-bases. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Vol. 2, pp. 623–628.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735–1780.
- Iyyer, M., Yih, W.-t., & Chang, M.-W. (2017). Search-based neural structured learning for sequential question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1, pp. 1821–1831.
- Jia, R., & Liang, P. (2016). Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12–22, Berlin, Germany.
- Kamp, H., & Reyle, U. (2013). *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, Vol. 42. Springer Science & Business Media.

- Kate, R. J., & Mooney, R. J. (2006). Using string-kernels for learning semantic parsers. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pp. 913–920. Association for Computational Linguistics.
- Kate, R. J., Wong, Y. W., & Mooney, R. J. (2005). Learning to Transform Natural to Formal Languages.. In *Proceedings for the 20th National Conference on Artificial Intelligence*, pp. 1062–1068, Pittsburgh, Pennsylvania.
- Kim, J., & Mooney, R. J. (2010). Generative alignment and semantic parsing for learning from ambiguous supervision. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 543–551. Association for Computational Linguistics.
- Kitano, H., Tambe, M., Stone, P., Veloso, M., Coradeschi, S., Osawa, E., Matsubara, H., Noda, I., & Asada, M. (1997). The robocup synthetic agent challenge 97. In *Robot Soccer World Cup*, pp. 62–73. Springer.
- Konstas, I., Iyer, S., Yatskar, M., Choi, Y., & Zettlemoyer, L. (2017). Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1, pp. 146–157.
- Kočiský, T., Melis, G., Grefenstette, E., Dyer, C., Ling, W., Blunsom, P., & Hermann, K. M. (2016). Semantic parsing with semi-supervised sequential autoencoders. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1078–1087, Austin, Texas.
- Krishnamurthy, J. (2016). Probabilistic models for learning a semantic parser lexicon. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 606–616.
- Krishnamurthy, J., Dasigi, P., & Gardner, M. (2017). Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1516–1526.
- Krishnamurthy, J., & Mitchell, T. (2012). Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 754–765, Jeju Island, Korea.
- Krishnamurthy, J., & Mitchell, T. M. (2015). Learning a compositional semantics for free-base with an open predicate vocabulary. *Transactions of the Association for Computational Linguistics*, 3, 257–270.
- Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2010a). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1223–1233, Cambridge, MA.
- Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2010b). Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings*

- of the 2010 Conference on Empirical Methods in Natural Language Processing, pp. 1223–1233, Cambridge, MA.
- Kwiatkowski, T., Choi, E., Artzi, Y., & Zettlemoyer, L. (2013). Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1545–1556, Seattle, Washington, USA.
- Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., & Steedman, M. (2011). Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pp. 1512–1523, Edinburgh, Scotland.
- Liang, P., Jordan, M., & Klein, D. (2011). Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 590–599, Portland, Oregon.
- Liang, P. S., Klein, D., Gillick, L., Cohen, J., Arsenault, L. K., Clausman, J., Pauls, A., & Hall, D. (2018). Data collection for a new conversational dialogue system.. US Patent App. 15/804,093.
- Long, R., Pasupat, P., & Liang, P. (2016). Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1456–1465, Berlin, Germany.
- Lu, W., Ng, H. T., Lee, W. S., & Zettlemoyer, L. S. (2008). A generative model for parsing natural language to meaning representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 783–792. Association for Computational Linguistics.
- Matuszek, C., Herbst, E., Zettlemoyer, L., & Fox, D. (2013). Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, pp. 403–415. Springer.
- Parikh, A., Täckström, O., Das, D., & Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2249–2255, Austin, Texas.
- Price, P. J. (1990). Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.
- Reddy, S., Lapata, M., & Steedman, M. (2014). Large-scale semantic parsing without question-answer pairs. *Transactions of the Association of Computational Linguistics*, 2(1), 377–392.
- Reddy, S., Täckström, O., Collins, M., Kwiatkowski, T., Das, D., Steedman, M., & Lapata, M. (2016). Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4, 127–140.
- Reddy, S., Täckström, O., Petrov, S., Steedman, M., & Lapata, M. (2017). Universal semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 89–101, Copenhagen, Denmark.

- Suhr, A., Iyer, S., & Artzi, Y. (2018). Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2238–2249.
- Wang, Y., Berant, J., Liang, P., et al. (2015). Building a semantic parser overnight.. In *Proceedings of the 53th Annual Meeting of the Association for Computational Linguistics*.
- Webber, B. L. (1978). A formal approach to discourse anaphora. Tech. rep., BOLT BERANEK AND NEWMAN INC CAMBRIDGE MA.
- Wen, T.-H., Gasic, M., Mrkšić, N., Su, P.-H., Vandyke, D., & Young, S. (2015). Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1711–1721, Austin, Texas.
- Winograd, T. (1972). Understanding natural language. *Cognitive psychology*, 3(1), 1–191.
- Wong, Y. W., & Mooney, R. (2006). Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pp. 439–446, New York City, USA.
- Wong, Y. W., & Mooney, R. (2007). Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 960–967, Prague, Czech Republic.
- Xu, X., Liu, C., & Song, D. (2017). SQLNet: Generating structured queries from natural language without reinforcement learning.. *abs/1711.04436*.
- Yin, P., & Neubig, G. (2017). A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 440–450, Vancouver, Canada.
- Yin, P., & Neubig, G. (2018). Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *EMNLP 2018*, 1, 7.
- Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pp. 1050–1055, Portland, Oregon.
- Zettlemoyer, L., & Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 678–687, Prague, Czech Republic.
- Zettlemoyer, L. S., & Collins, M. (2005a). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorical Grammars. In *Proceedings of 21st Conference in Uncertainty in Artificial Intelligence*, pp. 658–666, Edinburgh, Scotland.
- Zettlemoyer, L. S., & Collins, M. (2005b). Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorical Grammars. In *Proceedings of*

21st Conference in Uncertainty in Artificial Intelligence, pp. 658–666, Edinburgh, Scotland.

Zhong, V., Xiong, C., & Socher, R. (2017). Seq2SQL: Generating structured queries from natural language using reinforcement learning. *CoRR*, *abs/1709.00103*.